Secure Set Membership Using 3SAT*
(Extended Abstract)

Michael de Mare and Rebecca N. Wright

Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ 07030, USA

Abstract. A wide variety of powerful cryptographic tools have been built using RSA, Diffie-Hellman, and other similar assumptions as their basis. Computational security has been achieved relative to complexity assumptions about the computational difficulty of a variety of number theoretic problems. However, these problems are closely related, and it is likely that if any one of them turns out to be efficiently solvable with new mathematical advances or new kinds of computational devices, then similar techniques could be applicable to all of them. To provide greater diversity of security assumptions so that a break of one of them is less likely to yield a break of many or all of them, it is important to expand the body of computational problems on which security systems are based. Specifically, we suggest the use of hardness assumptions based on the complexity of logic problems, and in particular, we consider the well known Boolean 3SAT problem.

In this paper, we consider the use of the 3SAT problem to provide a cryptographic primitive, *secure set membership*. Secure set membership is a general problem for participants holding set elements to generate a representation of their set that can then be used to prove knowledge of set elements to others. Set membership protocols can be used, for example, for authentication problems such as digital credentials and some signature problems such as timestamping.

1 Introduction

The most popular computational foundation for cryptographic security is based on number-theoretic problems such as factoring, discrete logarithm, and elliptic logarithm [12,26,7]. These problems are all related, so if one is broken it is likely that they all will be broken [11]. Their security is not proven and is likely to either remain unproven or be broken. They are also vulnerable to quantum attacks [27]. It would be desirable to have many kinds of cryptographic primitives whose security is based on a wide array of unrelated assumptions. This would mean that if one system is compromised, they are not necessarily all compromised. In this paper, we present a system based on an alternative logic-based assumption that does not appear to be closely related to these other assumptions.

* This work was supported in part by the National Science Foundation under grant number CCR-0331584.

P. Ning, S. Qing, and N. Li (Eds.): ICICS 2006, LNCS 4307, pp. 452–468, 2006.

Specifically, we consider the use of the well known Boolean satisfiability problem to provide a very general cryptographic primitive, *secure set membership*. Secure set membership can be used to provide digital credentials, with or without identification, as well as for some signature problems such as timestamping. For example, consider a system for maintaining encrypted PINs for credit cards. Each credit card may have multiple PINs (for multiple users); any solution should hide the PINs in such a way that the system accepts valid PINs, but nobody can determine any valid PIN that he does not already know. If the system is used in a setting in which it is reasonable for the system to be able to determine which user it is talking to, then it is possible for the system to simply store hashes of all the valid PINs and compare a received hashed PIN with this list to determine if it is valid. This is an example of *credentials with identification*. However, if the users of the credit card do not want to identify themselves, or if the credit card issuer prefers to have users not identify themselves, except as a valid user of the credit card in question, when they make a purchase, then this results in the goal of *anonymous credentials*. For anonymous credentials, the user wants to prove that he has valid credentials without giving the credentials away.

In this paper, we provide a means for constructing a secure set membership system that can be used both for credentials with identification and for anonymous credentials. Secure set membership can be used as an alternative to digital signatures for some applications including timestamping [6]. We note that our system has the desirable property that each participant can choose her own set elements. In the setting of digital credentials, this allows participants to choose their credential values (rather than having them determined by a third party or as an output of a distributed credential generation algorithm), thus making the system suitable for use with credentials that are determined by user-chosen passwords or biometrics.

Our solution is based on the Boolean satisfiability problem (SAT), which has not previously been used for digital credentials. We are aware that the use of the problem of finding witnesses for 3SAT instances as a security assumption is unusual and the practice of basing cryptographic hardness on *NP*-completeness is shaky in general, because the worst case hardness required for *NP*-completeness does not say anything about most cases or the expected case. However, we think it is of interest nonetheless. First, algorithmic advances and new computing models threaten many of the commonly used cryptographic assumptions, such as the hardness of factoring. Secondly, SAT is perhaps one of the most studied *NP*-complete problems, and a fair bit is known about how to choose instances that appear to be hard. We discuss this further in Section 3.4 and Section 5 in the context of our proposed solution.

1.1 Related Work

The set membership problem was first addressed by Benaloh and de Mare with one-way accumulators in 1993 [6]. A number of schemes based on one-way accumulators were developed including schemes for digital credentials [10,3]. The schemes for credentials typically differ from the other schemes, which tend to

concentrate on the idea of a distributed signature, in several ways. These include central authorities in the credential scheme, as well as support of additional properties such as revocation. All these schemes depend on the difficulty of the RSA problem for their security.

Another approach to set membership is to use Merkle trees or similar tree-based methods to store the elements of the set [23,24,5]. In these methods, each participant retains a certificate and her own set element. In effect, each element of the set is signed by a central authority. However, these methods are either not storage-efficient or require more than a constant amount of time to check relative to the number of entries [28].

In a credential system, members of the credentialed group have, or are given, credentials that they can use to prove their membership in the set of authorized persons, without revealing which of the members they are. Biometric data may be used to prevent transferability of credentials, together with zero knowledge proofs of knowledge, for a group member to prove to a verifier that she holds a valid credential without revealing it. Anonymous credentials have been widely studied and solutions based on various cryptographic assumptions have been given (e.g. [9,10,1,21]).

Several approaches have been taken to digital credentials. Most of these approaches require a central authority (such as [6,10]), though some approaches based on one-way accumulators do not require a central authority. In contrast, our approach can work with or without a central authority. The combination of one-way accumulators and zero-knowledge proofs was introduced by Camenisch and Lysyanskaya [10]. Other credential systems allow revocation of anonymity such as a different system by Camenisch and Lysyanskaya [9].

Our work makes use of the assumed computational difficulty of finding satisfying assignments to certain kinds of satisfiable 3SAT instances. A related use of the hardness of SAT for achieving security has been recently proposed for hiding information in anomaly detection applications [14,15,16,13]. Their work is concerned with maintaining lists of information that, if compromised, will not compromise the larger system for applications such as intrusion detection. The central idea of our system is to represent an element of a set by an assignment to a set of variables, and the set of elements by a 3SAT instance that is satisfied by the corresponding assignments. In comparison, the work of Esponda et al. uses a SAT instance to represent a database; in their case, they represent the values *not* in the database by satisfying assignments.

We note that both our use of 3SAT and Esponda et al.'s use of SAT do not have the same difficulties as with earlier use of *NP*-complete problems for cryptography, such as the knapsack problem [25], because it is not necessary to embed a trapdoor to be used for operations such as decryption.

1.2 Our Contribution

Our contribution includes three protocols with applications to anonymous credentials, credentials with identification, accounts with multiple users, and digital timestamping.

Our protocols propose a solution to the set membership problem. Specifically, we provide a method for generating representations of sets of provided elements. We also provide a method of using a resulting representation to prove a particular element was in the set at the time the representation was generated, and a method of using the representation to show a party holds a valid set element without revealing the element itself. Our representations are random 3SAT instances of a particular form which accept the chosen witnesses. Theorem 1 shows that among 3SAT instances that accept the selected witnesses and have the selected number of clauses, the algorithm chooses one uniformly at random. The security of the scheme relies on the computational difficulty of finding satisfying assignments to such 3SAT instances. Our system has the following properties:

- It generates instances of 3SAT that are satisfied by a given set of strings.
- It generates any suitable instance of 3SAT with equal probability. This is shown in Theorem 1.
- In combination with zero knowledge proofs for 3SAT, it provides interactive proofs that can be used for anonymous credentials.
- Assuming the 3SAT instances generated are appropriately hard, it provides security against an attacker either finding a participant's element from the information needed to verify set membership or finding other bit strings that satisfy the set membership problem.

We define the set membership problem in Section 2. We present our system in Section 3. In Section 4, we discuss applications, including anonymous credentials and digital timestamping. We conclude with further discussion in Section 5.

2 Preliminaries

In this section, we define the secure set membership problem. A *secure set membership system* consists of two parts. First, the set must be established. Later, holders of set elements can prove their elements' set membership to others. Depending on the application, it may be desirable for the proof to reveal the set element or to keep it secret. Specifically, we have the following definitions.

Definition 1. *A* set establishment protocol *is a protocol carried out by some number m of* participants $P_1, \ldots, P_m$. *Each P_i holds as input* set element w_i. *The output of the protocol is a* set representation $T = T(w_1, \ldots, w_m)$.

Definition 2. *A* set membership protocol *is a protocol carried out by a participant P holding a set element w and a* verifier V *holding a set representation T. An honest verifier accepts if and only if the representation T was generated from a set of elements including w, even if P is cheating. The verifier learns w.*

Obviously, the set membership protocol is unsuitable for credential systems in which the set elements are reusable credentials, because it allows both V (and possibly eavesdroppers) to learn w and thereby to masquerade as P in the future to others. The protocol is also unsuitable for anonymous credential systems

unless further measures are taken because it allows V to distinguish between different provers because they have differing credentials. Fortunately, both of these difficulties can be eliminated by using a proof of possession protocol, defined below, instead of a set membership protocol.

Definition 3. *A* proof of possession protocol *is a protocol carried out by a participant P holding a set element w and a verifier V holding a set representation T. An honest verifier accepts if and only if the representation T was generated from a set of elements including w, even if P is cheating. The verifier V does not learn w, even if V is cheating.*

These definitions can be formalized according to the standard definitions of zero knowledge and simulatability.

In the sequel, we assume all participants are computationally bounded. In particular, our solutions depend on the computational infeasibility of finding witnesses for certain 3SAT expressions (also called *instances*). We discuss the validity of this assumption further in Section 3.4.

When we discuss a 3SAT instance, we pay attention to two parameters. These are the number, ℓ, of variables and the number, n, of clauses, also called the *size* of the instance. We also consider the *clause density* $\alpha = \frac{n}{\ell}$, which is an important parameter for determining the difficulty of a 3SAT instance [2].

In our solution, the elements of the set are interpreted as assignments to a set of variables, also called *witnesses*. We refer to a 3SAT instance that represents the set of elements as a *set representation* or, when clear from context, simply as a *set*.

3 Secure Set Membership

In this section, we describe our secure set membership protocols. We first describe in Section 3.1 a centralized process for a trusted party to establish a set representation for a set of given elements. In Section 3.2, we describe a distributed version of the set establishment protocol, which can be carried out by the participants holding set elements and does not require a centralized trusted party. In Section 3.3, we describe how to show set membership for elements of the established set. We discuss the security of our solutions in Section 3.4.

3.1 Centralized Set Establishment Protocol

Let $W = \{w_1, w_2, \cdots, w_m\}$ be a set of assignments to ℓ variables $V = \{v_1, \ldots, v_\ell\}$. Each w_i represents an individual element. The trusted party, say $\mathcal{T}$, generates a set representation for W—that is, a 3SAT instance satisfied by each $w_i \in W$. To do this, $\mathcal{T}$ repeatedly generates random clauses that are the conjunction of 3 literals over variables in V. He checks each clause he generates to determine whether it is satisfied by every $w_i \in W$. If there is some $w_i \in W$ that does not satisfy the clause, then $\mathcal{T}$ discards the clause and randomly selects a replacement clause which goes through the same test. Once n satisfied clauses are found, where n is a security parameter representing the desired size of the expression,

Input: A set of variable assignments $W = \{w_1, w_2, \cdots, w_m\}$, the number ℓ of variables to be used, and the target number n of clauses.
Output: A 3SAT instance satisfied by all $w \in W$.

While there are fewer than n clauses do:

1. Select three different random numbers $\{v_1, v_2, v_3\} \in \{1, \ldots, \ell\}$.
2. Select three random bits n_1, n_2, n_3. For each bit, if the bit is set, the corresponding random number is considered to be a negation of the variable.
3. If another clause has the same three numbers and corresponding negations discard v_1, v_2, v_3 and n_1, n_2, n_3 and return to Step 1.
4. For each w_j do
   If, for all $i \in \{1, 2, 3\}$ ((n_i is true and v_i is set in w_j) or (n_i is false and v_i is not set in w_j)) then delete $v_1, v_2, v_3, n_1, n_2, n_3$ and goto Step 1.
5. Add the clause represented by $\{(n_1, v_1), (n_2, v_2), (n_3, v_3)\}$ to the instance.

Algorithm 1. A centralized protocol for establishing a set

their conjunction forms the desired set representation T, which is output by $\mathcal{T}$. The complete algorithm is given in Algorithm 1.

Note that the output T is an instance of the 3SAT problem satisfied by the assignments that the participants have specified as elements. It may also be satisfied by some other unknown assignments. However, even if there are such spurious witnesses, that does not mean they are easy for an attacker to find. Nonetheless, it seems desirable to avoid having many such spurious witnesses. One can reduce the number of spurious witnesses by choosing a large n, because the probability of a given assignment satisfying a 3SAT instance decreases exponentially with the size of the instance. Specifically, n should be chosen to be large enough to satisfy three security criteria:

- The conjunction of the clauses should be satisfied by very few assignments that are not valid elements.
- The size of the conjunctive normal form (CNF) expression that is made by the clauses should be large enough that there is high probability that it is not an instance of SAT for which an efficient solution is known.
- The size of the CNF expression should be large enough that it can potentially be computationally infeasible to find satisfying assignments.

In general, this can be accomplished by choosing a suitably large number of variables and setting the clause density to a suitable value. The security of the scheme is discussed further in Section 3.4.

We now turn our attention to the computational complexity of this algorithm. We note that there is some chance that the algorithm might not even terminate, if there are not a sufficient number of available clauses that satisfy the given witnesses. However, if ℓ is chosen relatively large in comparison to m, and ℓ is sufficiently large compared to m and n, there should be a sufficient number of clauses that satisfy the witnesses. Further analysis or experiments are needed in order to determine appropriate values to use.

Assuming that there is a large number of clauses that satisfy the given witnesses, consider a particular witness representing one set element and consider a single randomly chosen 3SAT clause. There are three variables in a clause, all of which are given some assignment in the witness. Each variable in the clause can appear as a literal in either positive or negative form, so there are eight possible cases. Of these, seven are satisfied by the witness; it is only not satisfied (and therefore not accepted) in the case where none of the three literals is satisfied. Thus, the probability of a clause satisfying one witness is $\frac{7}{8}$. If there are m witnesses, then the probability of a clause satisfying all of them is $(\frac{7}{8})^m$. It follows that the expected number of tries required to generate a clause in the set representation is $(\frac{8}{7})^m$. It takes $O(\log \ell)$ bits to represent a clause and the clause must be checked against m witnesses, each of which can be done in constant time. Therefore, it takes $O(m \log \ell)$ time to test a clause to determine whether it is satisfied by all the witnesses.

In order to generate n clauses, it is necessary to find n distinct clauses that are satisfied by W. As each clause is found, it becomes slightly harder to find the next clause, as duplicates will sometimes be chosen. However, as long as n is very small relative to the total number of clauses that satisfy W, this has a negligible effect. If the probability that a random chosen clause passes both tests (satisies W and is not a duplicate) were fixed at $(\frac{7}{8})^m$, then the expected running time to generate a set representation would be $O(n((\frac{8}{7})^m)m \log \ell)$. We note that in cases where n is a significant fraction of the total number of clauses that satisfy W, then this would not be the case.

In practice, this means that it is only computationally efficient to generate an instance for at most up to around a hundred witnesses. A hundred witnesses leads to an expectation of 629,788 rejected clauses per accepted clause, easily doable with current computers. When the number of witnesses reaches a hundred and fifty, there is an expectation of about five hundred million rejected clauses for each accepted clause, probably infeasible for a typical modern computer when the number of clauses is considered.

3.2 Distributed Set Establishment Protocol

We now discuss the distributed protocol for establishing T, which is given in Algorithm 2. This algorithm works for *honest-but-curious* participants, who are assumed to follow their specified protocols. It also has some resilience against cheating participants; for example, cheating parties can cause an easy instance of 3SAT to be chosen, but in some cases the other participants can detect that this may be happening. At a high level, the protocol executes as follows: the participants locally generate local copies of the same random clause. Each determines if the clause is satisfied by her own witness and communicates this information to the others. If the clause is satisfied by all the witnesses, it is kept. Otherwise, it is discarded.

In order to protect the participants' witnesses from being disclosed, we use a verifiable secret-ballot election scheme by Benaloh [4]. The scheme is based on *homomorphic encryption* and *secret sharing*. It operates by designating some

participants as *tellers*. Participants give secret shares of their votes to the tellers. The tellers then use the homomorphic properties of the secret-sharing scheme to compute shares of the tally. They then collaborate to compute the actual tally and provide a proof to the participants that the tally was computed correctly.

In order to detect cheating of individual participants in our scheme, the tellers count the number of times that any participant votes "no" for any given clause. This can be accomplished without revealing the votes to the tellers by using the homomorphic property of the election scheme. The tellers maintain a running sum of each participant's votes and collaborate to determine that sum after a clause is chosen. If this sum exceeds a threshold value *maxreject*, then the instance is discarded and the protocol restarted from the beginning. Depending on the application setting for the protocol, it may be desirable to exclude participants who have exceeded the *maxreject* threshold some number of times from further participation. We note that even if a cheating participant succeeds in influencing the outcome of the protocol, she can neither learn another participant's witness nor cause another participant's witness to not satisfy the resulting 3SAT instance.

The goal is to choose *maxreject* high enough so that it detects cheating at levels that could lead to malicious participants being able to break the security of the result, but low enough so that it does not unnecessarily restart the protocol when no participants are cheating. As a somewhat arbitrary threshold, we suggest:

$$maxreject = \frac{\left(\frac{8}{7}\right)^m}{8} - \frac{n}{\log \frac{7}{8}} - \frac{2}{\log \frac{7}{8}},$$

which is derived as follows. As mentioned previously, the probability of a random clause satisfying a given witness is $\frac{7}{8}$. The first term of the formula for *maxreject* is the inverse of the probability of all the witnesses being satisfied for a single clause divided by the number of them that a single witness rejects. This is not sufficient to give a useful probability of an honest run not being rejected because there are n clauses yielding a probability that all n is satisfied of 2^{-n}. The second term of the formula for *maxreject* brings the probability of an honest run being rejected to 1/2 by being the solution to the equation: $(\frac{7}{8})^{f(n)} = 2^{-n}$. By adding $f(n)$ to the number of elections, we are dividing the probability by 2^{-n} for probabilities not approaching one. The third term further increases the probability that all terms are satisfied to $\frac{7}{8}$ by dividing the probability by $\frac{1}{4}$. Further analysis or experiments are needed in order to determine how effective this or any choice of *maxreject* is.

To ensure termination and also to provide some protection against multiple cheating participants colluding and "spreading out" their "no" votes in order not to individually exceed the *maxreject* threshold, it would also be a good idea to have a check in each iteration of the while loop that the loop has not been executed too many times, and to abort the protocol if this occurs.

In our set establishment protocol, the participants have a public shared source R of random or pseudorandom numbers. Using R, each participant generates a clause as the disjunction of three elements. Since the same random source is

Input: A set of variable assignments $W = \{w_1, w_2, \cdots, w_m\}$. Each w_i is known to participant P_i. All participants also know the number ℓ of variables to be used and the target number n of clauses, as well as a sufficiently long random string R.
Output: An instance of 3SAT that is satisfied by all participants' witnesses.

- set
$$maxreject = \frac{\left(\frac{8}{7}\right)^m}{8} - \frac{n}{\log \frac{7}{8}} - \frac{2}{\log \frac{7}{8}}$$
- While there are fewer than n clauses do:
  1. Using R, select three different variables v_1, v_2, v_3 and three flags n_1, n_2, n_3.
  2. Construct the clause where the flags denote the negation of variables.
  3. If the clause is equivalent to a clause already generated, discard it and return to Step 1.
  4. Hold a verifiable secret-ballot election (see [4]) using "yes" if the clause is satisfied by the witness and "no" otherwise. If the tally is unanimously "yes", then add the clause to the instance. Otherwise, delete it. Each teller should maintain a running sum of each participant's shares of votes.
  5. return to Step 1.
- Use the homomorphic property to compute the number of "no" votes for each participant. If one exceeds *maxreject*, discard all the clauses.

Algorithm 2. A distributed algorithm

used, all the participants generate the same clause. The participants hold a verifiable secret-ballot election. If the tally is unanimously "yes", the clause is kept; otherwise, it is rejected. If a participant votes "no", then the clause is discarded. This process is repeated until the target number n of clauses has been generated.

It is easy to verify that the output T is satisfied by all the inputs $w_1, \ldots, w_m$, so Algorithm 2 meets the definition of a set establishment protocol. Assuming that parties behave honestly, the expected number of tries to find a clause is $(\frac{8}{7})^m$ as in the centralized protocol of Section 3.1.

3.3 Set Membership

Our set representations lend themselves easily to both set membership and proof of possession protocols.

Set membership involves a participant P, who knows his element w, and a verifier V, who knows T. P wants to convince V that T was generated as a set representation that included the element w. In our case, then, P wants to convince V that w satisfies T.

A straightforward set membership protocol (in which V is allowed to communicate w to P, as per the definitions in Section 2), is for P to communicate w to V, who can then easily check in polynomial time whether w satisfies T. If it does, V accepts; otherwise, V rejects.

For proof of possession, it is important that the verifier never learns the credentials and cannot impersonate the prover. Fortunately, in our solution, it is not necessary to present the element to show set membership, but rather it is sufficient to show that one knows a satisfying string. This can be done with a zero knowledge proof. Assuming trapdoor one-way functions exist, then such zero knowledge proofs are possible for 3SAT using a generic construction that applies to any *NP*-complete problem [17]. Additionally, this can be made secure against quantum computers [29], in keeping with our motivation to avoid reliance on number-theoretic assumptions. If one is willing to rely on such assumptions, there are also simple examples of zero knowledge proofs for 3SAT that rely on factoring [8,4].

3.4 Security

The security of the set membership protocol and the proof of possession protocol depends on the difficulty of finding witnesses that satisfy a set representation T constructed by the set establishment protocol. We show below in Theorem 1 that the representation T is random among all instances of 3SAT with n clauses and ℓ variables satisfied by the specified assignments $W = \{w_1, \ldots, w_m\}$. The instance T may possibly be satisfied by some other assignments. That is, given a set of witnesses and a specified number of clauses, there is an equal probability that our algorithm produces any instance that is satisfied by the witnesses and has the proper number of variables and clauses. The probability that T is hard is the same as the probability that it is hard to find a witness for a random such instance of 3SAT. Unfortunately, it is not known what this probability is. (In fact, if $P = NP$, then the probability is zero.)

Our system rests on the assumption that a sufficiently large random instance of 3SAT satisfying a given set of witnesses and having an appropriately chosen clause density has a high probability of being hard to solve. If this assumption holds, then it is hard for anyone to find a witness which is not an element. It is also hard for a party who does not already know an element of T to find one. These two properties provide the security for both the set membership protocol and the proof of possession protocol. In particular, for the set membership protocol, the ability for an adversary to succeed in forging a witness without overhearing one is precisely the adversary's ability to determine a satisfying assignment to T, because this property can be exactly checked by the verifier. In the case of the proof of possession protocol, the security additionally relies on the soundness of the zero knowledge proof. An adversary who cannot find a valid witness has only negligible probability of convincing the verifier to accept.

Theorem 1. *Algorithms 1 and 2 generate with equal probability any* 3SAT *instance consisting of n different clauses that is satisfied by all the assignments in W.*

Proof (sketch). The same argument applies to both Algorithm 1 and Algorithm 2. This is because Algorithms 1 and 2 save or reject clauses for the same

reasons. The only difference is whether the checking is handled by the participants or by a centralized authority.

Consider the "random algorithm," which simply has a list of all the possible instances consisting of n distinct 3-literal clauses over ℓ variables that are satisfied by all $w \in W$ and selects one instance uniformly at random.

First, we show that our algorithm generates the same set of instances as the random algorithm. Suppose a possible 3Sat instance (in the random algorithm's list) cannot be generated by our algorithm. Then a clause in it must be rejected by our algorithm either because it is a duplicate or because some assignment does not satisfy the clause. It cannot be a duplicate, as this violates the requirement for the random algorithm's list that the clauses be distinct. If some assignment does not satisfy the clause, then no instance including that clause is satisfied by the assignment. Therefore, including it would violate the condition for the random algorithm's list that it must be satisfied by W. Hence, all instances in the list drawn on by the random algorithm are candidates for generation by our algorithm.

Conversely, suppose a 3Sat instance generated by our algorithm cannot be generated by the random algorithm. Then there are two possible reasons. The first is that there is a duplicate clause resulting in the number of unique clauses being less than n. This instance cannot be generated by our algorithm because the duplicate clause will be suppressed. The other possible reason is that it is not satisfied by one of the witnesses. In this case, one of the clauses is not satisfied by that witness (as the instance is a conjunction of the clauses). This clause will be rejected by our algorithm, so this instance cannot be generated. Therefore, the set of instances selected by the random algorithm is exactly the set of instances that our algorithm can generate. Call the size of this set N.

Finally, we show that our algorithm generates each instance with the same probability as the random algorithm. The random algorithm has probability $1/N$ of choosing each of the N instances that it can generate. Our algorithm also generates each of these instances with equal probability. To see this, note that in our algorithm, each clause has a constant probability depending on how many clauses have already been chosen. The product of a fixed number of constants is a constant. Therefore all of the instances have the same probability. It follows that, for our algorithm, each clause has probability $\frac{1}{N}$, as desired. ■

Theorem 1 states that, given ℓ and n, the system can generate any 3Sat instance of ℓ variables with n clauses that is satisfied by the specified witnesses. We make some observations and propose some heuristic recommendations for selecting the security parameters:

- Beyond a certain threshold, increasing the number of variables without increasing the number of clauses actually reduces security because there are not enough instantiations of the variables.
- Recall that the clause density of an instance is defined as $\alpha = \frac{n}{\ell}$. Alekhnovich and Ben-Sasson [2] show that if $\alpha \leq 1.63$, then the instance can be solved in linear time. They also demonstrate empirically that $\alpha < 2.5$ seems to be easy

to solve. We recommend taking $\alpha \geq 8$ (i.e., choosing $n \geq 8\ell$) for security. For example, $\ell = 128$ and $n = 1024$. If one is concerned about quantum attacks, then we suggest $\ell = 256$ and $n = 2048$ due to the quadratic advantage given by Grover's algorithm [18].

– A certain number of variables are trivial in any particular instance (i.e., because they either do not appear in positive form or in negative form, and therefore it is clear how to set them in a satisfying assignment). This can reduce the security of the system, by making it easier for an adversary to find satisfying assignments. Additionally, once the trivial variables are assigned, an adversary can then "remove" those clauses, potentially resulting in more trivial variables.

  If our instances were random among all 3SAT instances with n clauses and ℓ variables, then the expected number of trivial variables could be limited by taking the clause density sufficiently large. However, as noted before, our instances are random only among those 3SAT instances that are actually satisfied by the set W of witnesses. Further study is needed to determine how many trivial variables our instances are likely to have and whether this can be reduced.

The *phase boundary* of 3SAT is the clause density at which instances go abruptly from being mostly satisfiable to mostly unsatisfiable. The 3SAT decision problem—determining whether a 3SAT instance is satisfiable or not—is believed to be hardest when instances are just above the phase boundary [20]. However, our problem is a little different. Our set representation instances are always satisfiable (since they are specifically chosen to satisfy a particular set of witnesses). The problem at hand for an attacker is to find a satisfying assignment. We conjecture that the problem of finding satisfying assignments for instances that are known to be satisfiable gets harder as the probability of a random instance of the same parameter being satisfiable gets smaller—i.e., well above the phase boundary.

SATLIB contains resources for experimental research on SAT and 3SAT, including the results of competitions in solving random SAT instances. The literature on SATLIB suggests that progress has not been made on high clause density instances [20]. We also note that there is an optimization variant of the SAT problem called MAXSAT [20,19,22]. Specifically, it is possible to approximate 3SAT by finding assignments that satisfy most, but not all clauses. Known algorithms are polynomial time for finding a 7/8 assignment, but become exponential in the worst case when trying to do a full assignment.

Multiple cheating participants might collude to try to "spread out" their cheating rejections so that they can influence the outcome without exceeding *maxreject*. This can be compensated for by decreasing *maxreject* or by limiting the total number of rejections allowed cumulatively for all participants rather than for individual participants. However, this also increases the chance of "false positives," in which the protocol is restarted even without cheating behavior, so it is only likely to work well for a small number of colluding participants. It remains open to address other types of cheating and collusions.

4 Applications

There are a number of applications of the set membership problem, including credentials and document timestamping.

4.1 Digital Credentials

Our system applies to anonymous credentials in a fairly straightforward manner. The credentials are the elements. They are generated using either the centralized protocol or the distributed protocol and they are verified using the proof of possession protocol. In this way, the credentials are all generated at once and then the instance is distributed to the verifiers. Verifiers use the instance to anonymously determine whether a member is credentialed. If credentials with identification are desired, then the member can present his witness; the verifier can check that the witness satisfies the instance.

4.2 Accounts with Multiple Users

The system is also useful in situations where there need to be multiple authentication strings for a single account. An example is accounts with multiple users. Suppose there are three debit cards issued on one bank account and they all have the same number but each has a different PIN. The PINs can then be used as witnesses in constructing an instance. When a user wants to demonstrate that she is an authorized user of the account, she runs the proof of possession protocol using her PIN. This way, joint holders of an account can access the account without giving away their PINs (which might also be used for other accounts that are not shared).

Other applications of multiple user accounts include the use of RFID tags as witnesses in an access control system based on proximity sensors and other access control situations where it is not desirable to uniquely identify the user.

4.3 Document Timestamping

Document timestamping [5,6] may require a little more explanation. In document timestamping applications, we think of the distributed protocol as a distributed signature. All the parties participating in the protocol are attesting that one of their number knew each witness at the time the protocol was run by accepting the set that results from the protocol. It would not be possible for the protocol participant to execute the protocol and then choose a satisfying witness at some later date.

The timestamping system proceeds in rounds. All documents submitted during the same round are considered to be simultaneous, like patent applications arriving at the patent office on the same day. Each participant's witness is a hash of the document(s) she would like to timestamp. The distributed protocol is run, and everyone remembers the round's set, which is the timestamp. The parties may jointly publish it if they wish to allow anyone to verify a timestamp.

To verify that a document was submitted during a given round, the verifier merely needs to run the set membership protocol. The security of this system does not depend on computational security, in that if a cheating prover wishes to make his specific document appear to be timestamped and it does not satisfy the 3SAT instance, there is nothing he can do to change that. (We note, though, that in most practical settings, the adversary may be able to change his document in ways that do not affect its meaning, but do affect its encoding into a bit string, so this guarantee is not absolute.)

Digital timestamping can be used for intellectual property disputes, among other applications. In the intellectual property application, a consortium generates a timestamp with each company using a hash of the hashes of all of its documents. Each company retains the daily timestamp and publishes it for other interested parties. In a patent dispute, for instance, a party can get all the other honest participants to attest to its possession of a document on or before a certain date. This could also be used to prevent backdating in stock or other business transactions.

5 Discussion

We have presented a general solution to the set membership problem whose security depends on the difficulty of finding witnesses to random 3SAT instances satisfying a given set of witnesses. We have also presented applications to access control, digital credentials, and timestamping. We have shown a distributed protocol for establishing a set.

A strong justification for considering security based on 3SAT is the increased worry that advances in conventional or quantum computing may one day yield efficient algorithms for problems such as factoring and discrete logarithms typically used as a source of hardness in cryptography. It is therefore important to investigate cryptographic algorithms based on alternate (plausible) hardness assumptions to provide resilience against "breaking" of any one assumption or class of assumptions.

Further work includes analysis and experiments to determine the probability distribution of the output of our set establishment protocol with respect to all 3SAT instances. In particular, it should be investigated experimentally whether a randomly generated instance of 3SAT of size n that is satisfied by the chosen witnesses falls into one of the patterns whose solution is known to be easy, as well as determining whether all such instances can be specifically avoided.

Our protocol can be used for digital credentials including anonymous credentials, timestamping, and other set membership applications. It can also be used for applications where multiple users share an account. These include some access control and financial applications. For set membership applications like timestamping, the set representation can be thought of as a distributed signature. It can be proven to any honest participant or observer using the set membership protocol that a document was used for inclusion in the set. These applications have broad applicability to problems in cryptography and security.

The advantages of this method over one-way accumulators include not needing to remember a second string and not being dependent on the factoring problem [6].

As previously discussed, the expected number of clauses that must be tried to generate a clause in the set representation is $(\frac{8}{7})^m$, where m is the number of witnesses to be represented. We note that this probability depends on the number of elements and is independent of n and ℓ. In contrast, the security of the system is based on the adversary's difficulty of finding an element as a function of n and ℓ, so it may be possible to limit m so as to have efficient solutions for the participants without making the adversary's task solvable. As described earlier in Section 3.1, we believe that one hundred witnesses can be dealt with easily, but that as the number of witnesses begins to reach one hundred fifty, it becomes infeasible to generate an instance. For the distributed protocol, the limits may be slightly lower to compensate for communication overhead. This can be countered by replacing 3SAT with k-SAT where k is $\Theta(m)$. This eliminates the exponential complexity for the participants. However, in this case, it is necessary to make α significantly greater than the phase boundary for k-SAT. It is an open problem to determine the phase boundary of k-SAT for $k > 3$.

The space complexity for a set based on 3SAT is $\Theta(\ell \log \ell)$. For instance, a system with 128 variables requires 1024 clauses. Altogether, this requires three kilobytes of storage. This space complexity is independent of the number of set elements. However, if k-SAT is used instead of 3SAT, then the space complexity grows both in the number of bits required to represent a clause and in the number of clauses required to be above the phase boundary.

Directions for future research include developing a better understanding of the expected hardness of the 3SAT instances generated by our algorithm and extending the distributed set establishment protocol to efficiently handle general malicious behavior.

Acknowledgements

We thank David Evans and the anonymous reviewers for their helpful comments.

References

1. Alessandro Acquisiti. Anonymous credentials through acid mixing, 2003. Unpublished manuscript.
2. Mikhail Alekhnovich and Eli Ben-Sasson. Linear upper bounds for random walk on small density random 3-CNFs. In *Proceedings of the 44th Annual IEEE Symposium on the Foundations of Computer Science*, 2003.
3. Niko Baric and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Advances in Cryptology – Eurocrypt '97*, volume 1233 of *Lecture Notes in Computer Science*. Springer, 1997.
4. Josh Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University Department of Computer Science, September 1987.
5. Josh Benaloh and Michael de Mare. Efficient broadcast time-stamping. Technical Report TR-MCS-91-1, Clarkson University Department of Mathematics and Computer Science, 1991.

6. Josh Benaloh and Michael de Mare. One-way accumulators: A decentralized approach to digital signatures. In *Advances In Cryptology – Eurocrypt '93*, volume 765 of *Lecture Notes in Computer Science*, pages 274–285. Springer, 1994.
7. Dan Boneh and Matthew Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.
8. G. Brassard and C. Crepeau. Zero-knowledge simulation of boolean circuits. In *Advances in Cryptography – Crypto '86*, volume 263 of *Lecture Notes in Computer Science*, pages 223–233. Springer, 1987.
9. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology – Eurocrypt 2001*, volume 2045 of *Lecture Notes in Computer Science*. Springer, 2001.
10. Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in Cryptology – Crypto 2002*, volume 2442 of *Lecture Notes in Computer Science*. Springer, 2002.
11. Michael de Mare. An analysis of certain cryptosystems and related mathematics. Master's thesis, State University of New York Institute of Technology, Dec. 2004.
12. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, Nov. 1976.
13. F. Esponda. *Negative Representations of Information*. PhD thesis, University of New Mexico, 2005.
14. Fernando Esponda, Elena S. Ackley, Stephanie Forrest, and Paul Helman. On-line negative databases. In *Proceedings of the 3rd International Conference on Artificial Immune Systems (ICARIS)*, pages 175–188. Springer-Verlag, Sep. 2004.
15. Fernando Esponda, Stephanie Forrest, and Paul Helman. Enhancing privacy through negative representations of data. Technical report, University of New Mexico, 2004.
16. Fernando Esponda, Stephanie Forrest, and Paul Helman. Information hiding through negative representations of data. Technical report, University of New Mexico, 2004.
17. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP statements in zero-knowledge and a methodology of cryptographic protocol design. In *Advances in Cryptology – Crypto '86*, volume 263 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 1987.
18. Lov Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 212–219. ACM Press, 1996.
19. Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.
20. Holger H. Hoos and Thomas Stützle. SATLIB: An online resource for research on SAT. In *SAT 2000*, pages 283–292. IOS Press, 2000. `http://www.satlib.org`.
21. Russell Impagliazzo and Sara Miner. Anonymous credentials with biometrically-enforced non-transferability. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, pages 60–71. ACM, 2003.
22. Howard J. Karloff and Uri Zwick. A 7/8-approximation algorithm for MAX 3SAT? In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 406–415. IEEE Computer Society, 1997.
23. Ralph C. Merkle. *Secrecy, authentication, and public key systems*. UMI Research Press, 1982.

24. Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology – Crypto '87*, volume 293 of *Lecture Notes in Computer Science*. Springer, 1988.
25. A. M. Odlyzko. The rise and fall of the knapsack cryptosystems. In *PSAM: Proceedings of the 42nd Symposium in Applied Mathematics*, pages 75–88, 1990.
26. R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, Feb. 1978.
27. P. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE, 1994.
28. Michael Szydlo. Merkle tree traversal in log space and time. In *Advances in Cryptology – Eurocrypt 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 541–554, 2004.
29. John Watrous. Zero knowledge against quantum attacks. In *STOC '06 – 38th Annual ACM Symposium on Theory of Computing*, pages 296–315. ACM, 2006.